UNIVERSIDADE FEDERAL DE VIÇOSA - UFV - CAMPUS FLORESTAL

META-HEURÍSTICAS

Trabalho Prático 1

Aymê Faustino dos Santos - 4704

Henrique Alves Campos - 4231

Pedro Augusto Martins Pereira - 4692

Florestal - MG

2025

Sumário

1. Introdução

Neste trabalho prático foram implementados dois algoritmos de otimização para resolver dois problemas de minimização, cada um em dois intervalos diferentes. O primeiro algoritmo é baseado em VNS (Variable Neighborhood Search), enquanto o segundo é baseado em Simulated Annealing (SA). O objetivo foi explorar e comparar o desempenho deles na resolução das duas funções objetivo.

Os algoritmos foram implementados na linguagem de programação Python e organizados em um Jupyter Notebook para facilitar a visualização e a execução dos códigos. Para avaliar o desempenho dos algoritmos, cada um deles foi executado 30 vezes de forma independente para cada função objetivo em cada intervalo. Isso nos permitiu coletar dados estatísticos, como o valor mínimo, valor médio e desvio padrão do valor final da função objetivo em cada execução. Com essas estatísticas é possível avaliar a eficácia e robustez dos algoritmos em diferentes cenários de otimização.

Ao longo deste relatório, as funções objetivo e os intervalos em que foram otimizadas serão detalhadas, a implementação dos algoritmos será descrita e uma análise estatística abrangente dos resultados obtidos será apresentada.

2. Funções Objetivo

Um dos componentes básicos de um problema de otimização é a função objetivo. Esta função depende das variáveis de decisão, que são as variáveis cujos valores são procurados, e do domínio da otimização, ou espaço de busca, que é o intervalo onde os valores das variáveis devem ser encontrados.

Para modelar matematicamente um problema de otimização é necessário definir como as variáveis de decisão serão representadas. Neste trabalho, como as duas funções objetivo possuem números diferentes de variáveis de decisão em um espaço contínuo, foi utilizada a estrutura de dados de vetor.

Com o objetivo de entender algumas características das funções objetivo e facilitar a análise do comportamento delas dentro dos intervalos definidos, foi utilizada a biblioteca `Matplotlib` para criar gráficos 3D das superfícies das funções. Como cada função objetivo foi otimizada em dois intervalos diferentes, os gráficos foram gerados separadamente para cada intervalo de forma a observar as variações e identificar possíveis padrões de comportamento, como a existência de mínimos locais e a forma geral da superfície.

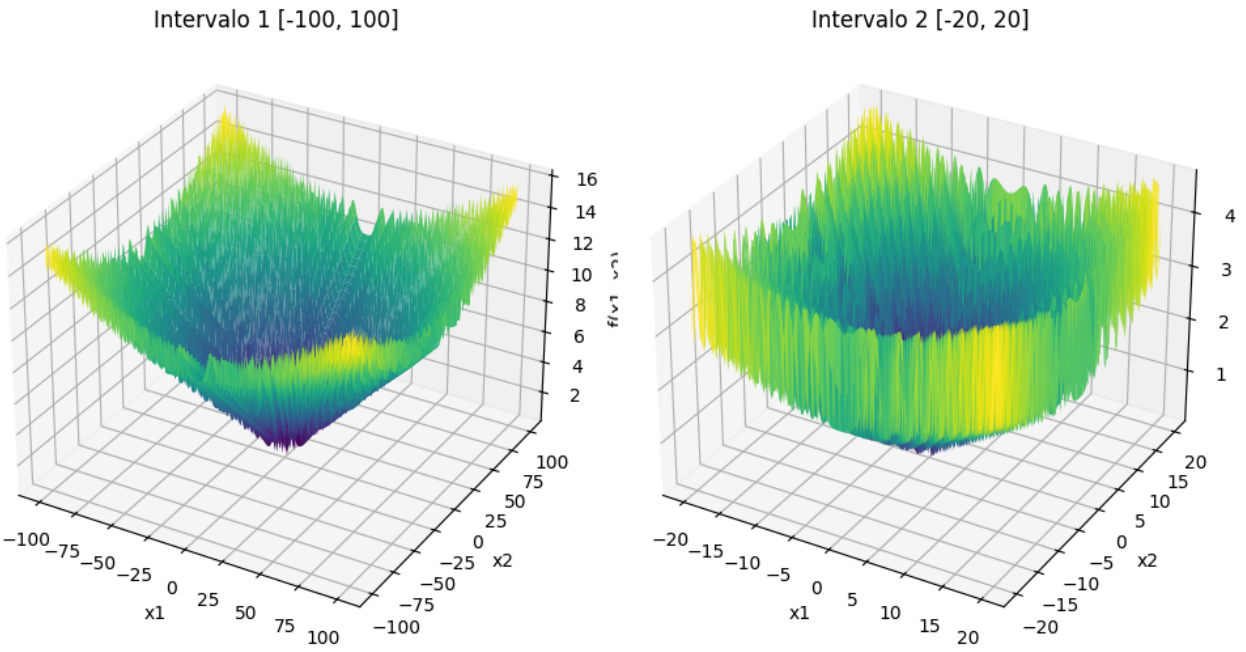
2.1. Função Objetivo 1

$$f(\mathbf{x}) = 1 - cos(2\pi\sqrt{\sum_{i=1}^{d} x_i^2}) + 0.1\sqrt{\sum_{i=1}^{d} x_i^2}$$

2.1.1 Intervalos

$a)\ -100 \leq x1,\ x2 \leq 100$
$b) -20 \leq x1,\ x2 \leq 20$

2.1.2 Superfície



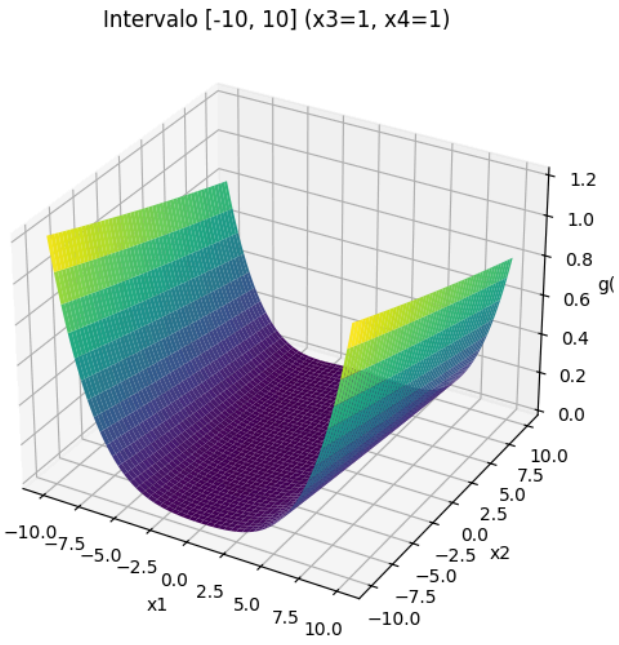
2.2. Função Objetivo 2

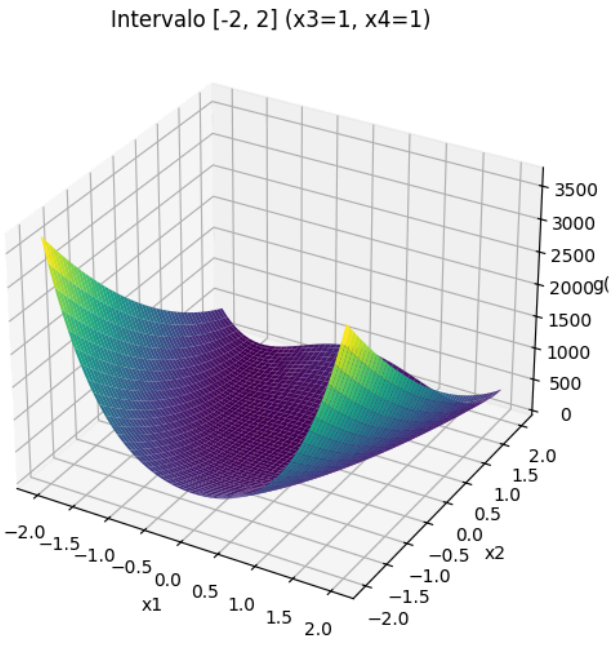
$$f(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$$

2.2.1 Intervalos

$a)\ -10 \leq x1, x2, x3, x4 \leq 10$
$b) -2 \leq x1, x2, x3, x4 \leq 2$

2.2.2 Superfície





Como a função f(x) depende de quatro variáveis, foi necessário fixar x_3 e x_4 para permitir a visualização em 3D. Assim, foi possível analisar o comportamento da função variando apenas x_1 e x_2.

3. Algoritmos

3.1. Variable Neighborhood Search (VNS)

O algoritmo Variable Neighborhood Search (VNS) é uma metaheurística baseada na ideia de explorar sistematicamente diferentes estruturas de vizinhança para escapar de ótimos locais e encontrar melhores soluções em problemas de otimização. O VNS alterna entre fases de perturbação (exploração) e busca local (intensificação), no nosso caso utilizando o algoritmo Hill Climbing como método de melhoria local. A implementação do VNS foi guiada pelo pseudocódigo apresentado abaixo:

```
s ← solucaoInicialAleatoria()    // Solução inicial aleatória
melhor ← s
k_max ← número máximo de vizinhanças
iter ← 0
enquanto iter < max_iter faça
    k ← 1
    enquanto k ≤ k_max faça
        s' ← perturbar(s)                    // Exploração: perturbando a solução corrente
        s' ← hillClimbing(s')                // Intensificação (busca local) com Hill Climbing
        se qualidade(s') < qualidade(melhor) então
            melhor ← s'
            s ← s'
              k ← 1                                  // Reinicia vizinhança após encontrar melhor
solução
        senão
            k ← k + 1
    iter ← iter + 1
retorne melhor
```

3.1.1 Solução Inicial

Para obter uma solução inicial, foi utilizada a estratégia de gerar um vetor com valores aleatórios dentro do intervalo de cada variável.

3.1.2 Método de Busca Local

Como método de busca local, foi utilizado o HILL Climbing que é um método que usa um procedimento de melhoria iterativa, ou seja, a cada iteração ele tenta melhorar a solução corrente. Assim como no VNS, a implementação do algoritmo foi guiada pelo pseudocódigo, descrito abaixo, apresentado em sala de aula.

```
s // solução candidata inicial
repita
    r = pertubar(copia(s)) // faz uma perturbação pequena na solução corrente
    se qualidade(r) < qualidade(s) então
        s = r // se melhora atualiza solução corrente
até condição de parada
retorne s
```

3.1.2.1 Solução Inicial

A solução inicial do Hill Climbing é definida dentro do próprio VNS.

3.1.2.2 Perturbar Solução Corrente

Perturbar a solução corrente no Hill Climbing envolve a adição de um pequeno valor aleatório, conhecido como ruído, a cada variável de decisão. Para essa etapa foi definida uma função, cujo pseudocódigo está abaixo, que recebe o intervalo de porcentagem do ruído como parâmetro, permitindo sua fácil adaptação para diferentes cenários, como para perturbações maiores ou menores. Para as pequenas perturbações, definimos o intervalo de ruído entre 3% e 5%.

```
min = valor mínimo desejado para cada elemento do vetor
max = valor máximo desejado para cada elemento do vetor

min_ruido = porcentagem mínima do ruído
max_ruido = porcentagem máxima do ruído

v = quantidade de variáveis de decisão
vetor[v] = solução a ser perturbada

for i de 1 até v faça --> para cada elemento do vetor de variáveis
    ruido = número aleatório [min_ruido, max_ruido]
       perturbacao = vetor[i] + ruido * (número aleatório [min[i], max[i]] -
vetor[i])

    se perturbacao < min[i] ou perturbacao > max[i] então
        enquanto perturbacao < min[i] ou perturbacao > max[i] faça
            ruido = número aleatório [min_ruido, max_ruido]
              perturbacao = vetor[i] + ruido * (número aleatório [min[i], max[i]] -
vetor[i])
    adicionar perturbacao ao vetor
retornar vetor como array numpy
```

3.1.2.3 Critério de Parada

O critério de parada utilizado na implementação do Hill Climbing é realizar 5 iterações sem melhoria da solução atual

3.1.3 Definição da Vizinhança

A definição de vizinhança é realizada por meio de uma perturbação aleatória da solução corrente. A cada iteração, um novo candidato à solução é gerado somando-se à solução atual um vetor de números aleatórios, com valores uniformemente distribuídos no intervalo [−1,1] para cada variável de decisão. Dessa forma, a vizinhança é composta por soluções próximas, obtidas através de pequenas variações nas variáveis.

3.1.4 Critério de Aceitação

Para determinar qual ótimo local será mantido para a próxima iteração, é necessário estabelecer um critério de aceitação. Esse critério permite ao algoritmo equilibrar entre intensificação e diversificação: aceitar sempre o melhor ótimo favorece a intensificação, enquanto aceitar sempre o ótimo local mais recente favorece a exploração. Nesse sentido, na implementação feita favorecemos a intensificação pois sempre buscamos a melhor solução.

3.1.5 Critério de Parada

O critério de parada é definido por um número máximo de 100 iterações.

3.2. Simulated Annealing (SA)

O algoritmo **Simulated Annealing (SA)** é uma metaheurística inspirada no processo de recozimento utilizado na metalurgia. Seu objetivo é escapar de ótimos locais, explorando a solução de maneira aleatória enquanto a temperatura diminui progressivamente.

3.2.1 Solução Inicial

A solução inicial é gerada aleatoriamente dentro dos limites definidos para cada variável de decisão. Este processo inicial é importante para garantir que o algoritmo explore diferentes regiões do espaço de soluções.

- **Implementação:** Uma função gera um vetor de variáveis de decisão dentro do intervalo definido para cada variável.

```
def solucao_inicial(self):
    return np.random.uniform(self.limite_inf, self.limite_sup, self.d)
```

3.2.2 Temperatura Inicial

.A temperatura inicial, denotada por T_inicial, define o nível de exploração do algoritmo nas primeiras iterações. Inicialmente, com a temperatura alta, o algoritmo permite a aceitação de soluções piores, o que ajuda na exploração do espaço de soluções.

- **Implementação:** A temperatura começa com um valor alto, no nosso caso decidimos usar o valor 1000 e é gradualmente reduzida ao longo das iterações.

```
def executar(self, n_execucoes = 30, T_inicial = 1000, T_final = 0.01, alpha = 0.95, iteracoes_T = 150, passo_max = None):
    if passo_max is None:
        passo_max = (self.limite_sup - self.limite_inf) * 0.1 # Usa 10% como o padrão
```

3.2.3 Quantidade de Transições em uma Temperatura

Em cada temperatura, o algoritmo realiza um número definido de transições ou iterações, buscando vizinhos da solução corrente. Durante cada transição, uma nova solução é gerada e aceita com base em um critério probabilístico.

- **Implementação:** A cada iteração, uma solução vizinha é gerada pela perturbação da solução corrente, e essa solução é aceita com uma probabilidade dependente da diferença de valor entre a nova solução e a solução corrente.

```
def gerar_vizinho(self, x_atual, passo_max):
    perturbacao = np.random.uniform(-passo_max, passo_max, self.d)
    x_vizinho = x_atual + perturbacao
    return np.clip(x_vizinho, self.limite_inf, self.limite_sup)
```

3.2.4 Controle da Temperatura

A temperatura é controlada por um fator de resfriamento, que reduz progressivamente a temperatura à medida que o algoritmo avança. O fator de resfriamento é um valor alpha (geralmente entre 0.9 e 0.99) que multiplica a temperatura a cada iteração.

- **Implementação:** A cada iteração, a temperatura é atualizada com o valor de T =* alpha, onde T é a temperatura atual e alpha é o fator de resfriamento.

```
    T *= alpha  # Resfriamento
```

3.2.5 Perturbar Solução Corrente

A perturbação da solução corrente é uma das etapas fundamentais do algoritmo, sendo feita de forma aleatória dentro do espaço de soluções. A perturbação pode ser configurada com um limite máximo de variação (passo), o que ajuda a controlar a exploração da solução.

- **Implementação:** Uma solução vizinha é gerada ao adicionar um ruído aleatório à solução atual. Esse ruído é calculado como uma variação aleatória proporcional à solução corrente.

3.2.6 Critério de Parada

O critério de parada é utilizado para encerrar a execução do algoritmo. Pode ser baseado em várias condições, como um número máximo de iterações ou quando a temperatura atinge um valor mínimo predefinido.

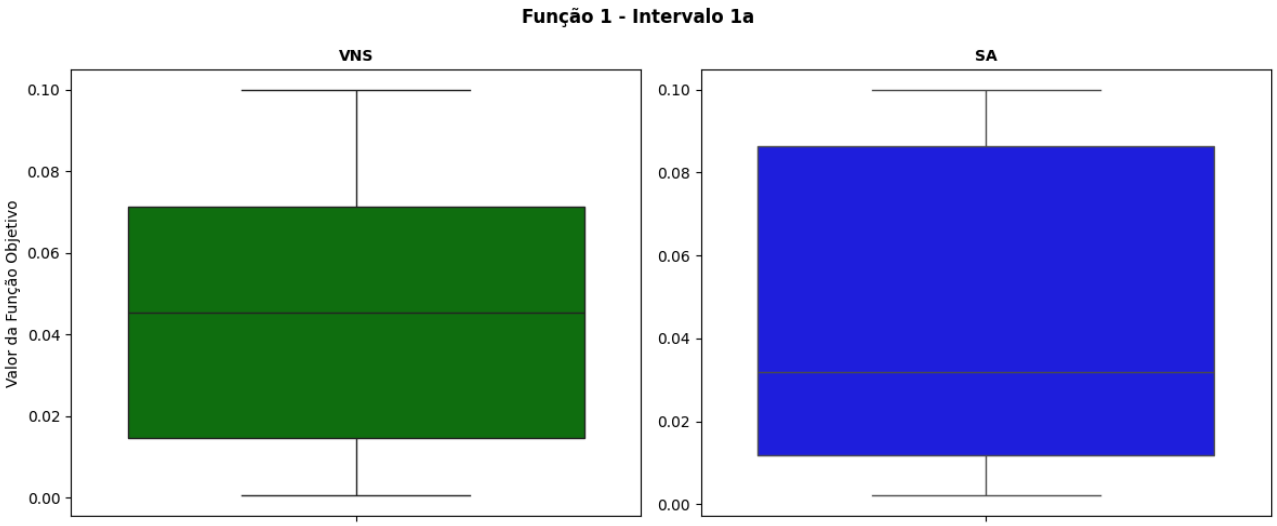
- **Implementação:** O critério de parada é definido pela temperatura final T_final, e o algoritmo para quando a temperatura atinge esse valor. Uma alternativa seria também parar após um número máximo de iterações.

4. Resultados

4.1. Problema com Função Objetivo 1 com Intervalo A) para as variáveis de decisão

Algoritmo	Mínimo	Máximo	Média	Desvio-Padrão
VNS	**0.000437**	**0.099874**	**0.045134**	**0.031531**
SA	**0.002091**	**0.099893**	**0.046749**	**0.037087**

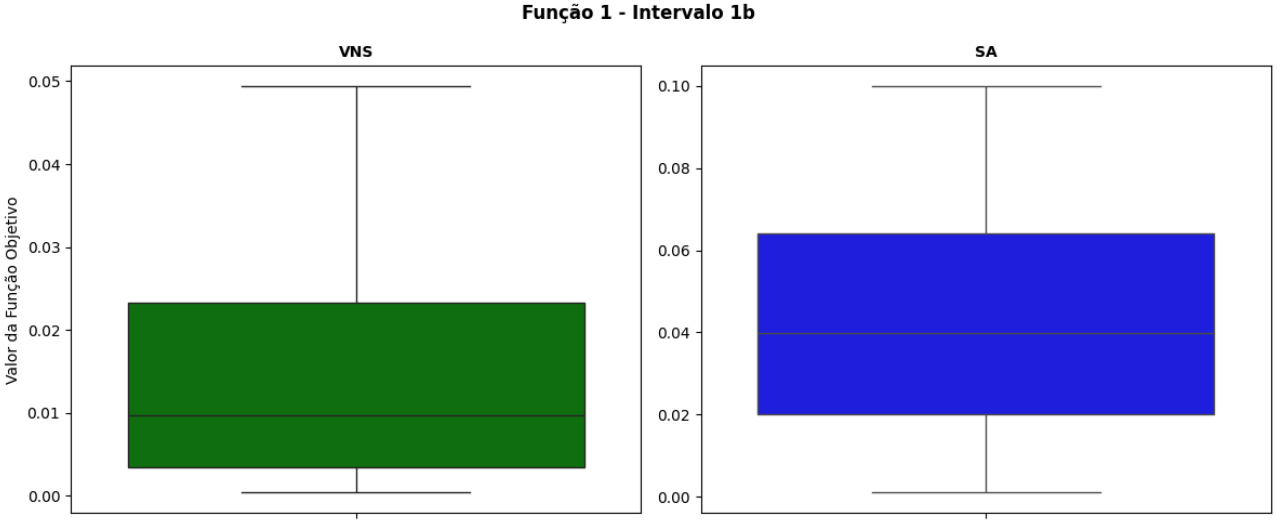
Os valores das variáveis de decisão para a melhor solução encontrada pelo VNS são X1 = 0.002488 e X2 = 0.001305. Já para o SA, os valores das variáveis de decisão para a melhor solução encontrada são e X1 = 0.006743 e X2 = 0.004429.



4.2. Problema com Função Objetivo 1 com Intervalo B) para as variáveis de decisão

Algoritmo	Mínimo	Máximo	Média	Desvio-Padrão
VNS	**0.000386**	**0.049390**	**0.013619**	**0.012682**
SA	**0.001045**	**0.099875**	**0.045864**	**0.030736**

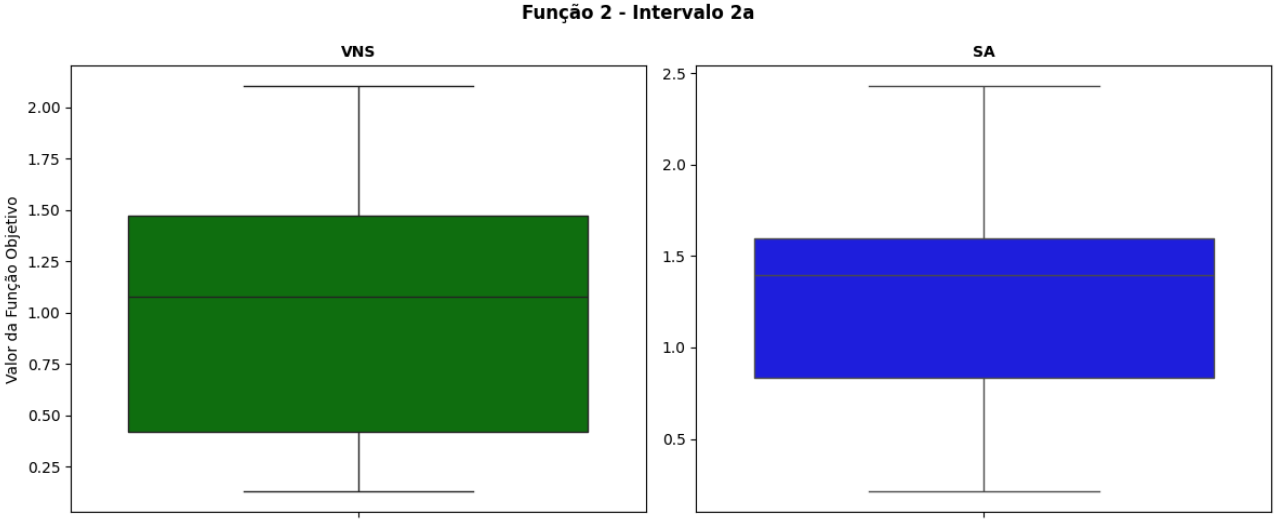
Os valores das variáveis de decisão para a melhor solução encontrada pelo VNS são X1 = 0.001123 e X2 = 0.002304. Já para o SA, os valores das variáveis de decisão para a melhor solução encontrada são X1 = -0.003954 e X2 = 0.003334.



4.3. Problema com Função Objetivo 2 com Intervalo A) para as variáveis de decisão

Algoritmo	Mínimo	Máximo	Média	Desvio-Padrão
VNS	**0.127442**	**2.103292**	**0.992171**	**0.580095**
SA	**0.210874**	**2.430890**	**1.291647**	**0.623434**

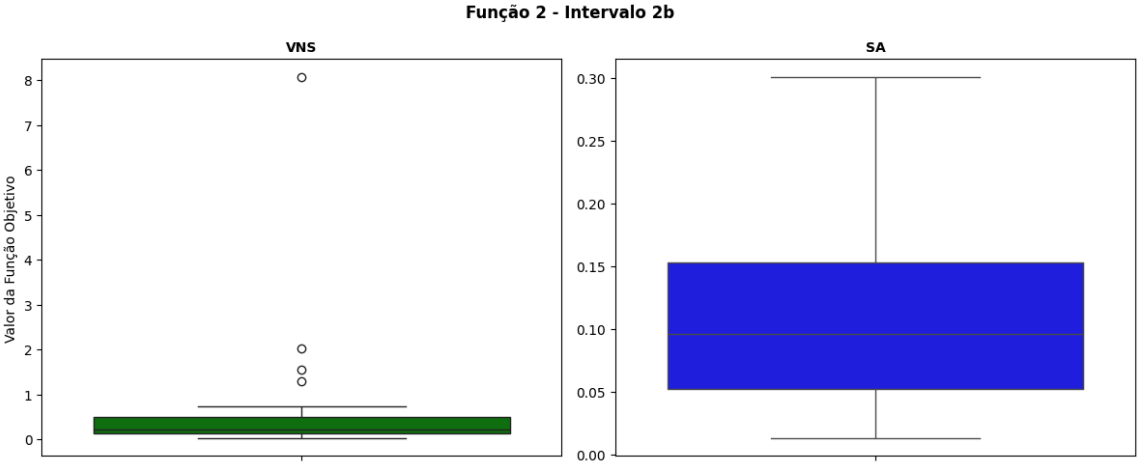
Os valores das variáveis de decisão para a melhor solução encontrada pelo VNS são X1 = 1.078830, X2 = 1.152707, X3 = 0.974696 e X4 = 0.930709. Já para o SA, os valores das variáveis de decisão para a melhor solução encontrada são X1 = 1.139132, X2 = 1.325695 X3 = 0.802588 e X4 = 0.633864.



4.4. Problema com Função Objetivo 2 com Intervalo B) para as variáveis de decisão

Algoritmo	Mínimo	Máximo	Média	Desvio-Padrão
VNS	**0.027882**	**8.062830**	**0.642469**	**1.45206**
SA	**0.012862**	**0.300322**	**0.106381**	**0.06498**

Os valores das variáveis de decisão para a melhor solução encontrada pelo VNS são X1 = 1.010472, X2 = 1.152707, X3 = 0.974696 e X4 = 0.930709. Já para o SA, os valores das variáveis de decisão para a melhor solução encontrada são X1 = 0.986153, X2 = 0.969064, X3 = 1.031186 e X4 = 1.058472



5. Conclusão

Este trabalho teve como objetivo aplicar e comparar duas meta-heurísticas — o VNS (Variable Neighborhood Search) com Hill Climbing como busca local, e o Simulated Annealing (SA) — na minimização de duas funções objetivo distintas, testadas sob diferentes intervalos de busca.

Função 1

A Função 1 é uma função multimodal baseada na distância euclidiana ao ponto de origem. Ao analisarmos os resultados nos dois intervalos definidos, notamos que o VNS teve desempenho superior ao SA em todos os aspectos

- No intervalo [-100, 100], o VNS obteve um valor mínimo de 0.000437, enquanto o SA encontrou 0.002091. A média dos resultados do VNS foi 0.045134 contra 0.046749 do SA, com um desvio padrão menor (0.0315 contra 0.0370), indicando maior estabilidade das soluções.

- No intervalo mais restrito [-20, 20], a superioridade do VNS se torna ainda mais evidente. O valor mínimo foi 0.000386 para o VNS, contra 0.001045 do SA. A média também foi significativamente inferior (0.0136 vs. 0.0458) e com menor dispersão, reforçando a consistência do VNS mesmo em espaços de busca reduzidos.

Além disso, os valores de X1 e X2 associados aos mínimos mostram que ambas as meta-heurísticas convergiram para pontos próximos da origem, como esperado dado o comportamento da função.

Função 2

A Função 2 apresenta uma estrutura mais complexa e altamente não linear, com múltiplas variáveis e acoplamentos entre elas. Os resultados nos dois intervalos testados revelaram diferenças mais sutis entre os algoritmos:

- No intervalo mais amplo [-10, 10], o VNS obteve novamente o melhor valor mínimo (0.127) frente ao SA (0.210). A média dos resultados do VNS foi 0.992, inferior à média do SA (1.291), e com desvio padrão também menor (0.580 contra 0.623), indicando desempenho mais consistente. Os valores de X1 a X4 obtidos mostram que as soluções convergiram para valores próximos a 1, o que está de acordo com os mínimos esperados da função.

- No intervalo mais restrito [-2, 2], os resultados foram mais interessantes: o SA encontrou um valor mínimo inferior ao do VNS (0.0128 contra 0.0278), e também obteve a menor média geral (0.106 contra 0.642). Neste cenário, o SA superou o VNS tanto em qualidade das soluções quanto em estabilidade (menor desvio padrão). Isso sugere que o SA foi mais eficiente em explorar intensamente um espaço de busca menor e mais concentrado.

Os resultados obtidos indicam que o VNS foi consistentemente mais robusto, especialmente em espaços de busca amplos e para funções mais suaves. No entanto, o SA demonstrou maior eficácia em cenários com intervalo de busca restrito, possivelmente devido à sua capacidade de escapar de ótimos locais por meio da aceitação de soluções piores com base em uma política probabilística.

Portanto, a escolha entre VNS e SA pode depender do problema em questão e do domínio de busca considerado. O VNS tende a oferecer soluções mais estáveis e próximas do ótimo global em problemas mais amplos, enquanto o SA pode ser uma boa escolha em domínios mais estreitos e para funções com muitos mínimos locais.

6. Referências

Slides da disciplina de Meta-Heurísticas (CCF-480) ministrada pelo Professor Marcus Henrique Soares Mendes, UFV - Campus Florestal.